Starting with this homework, groups of up to three students may submit a single solution for each numbered problem. Every student in the group receives the same grade. Groups can be different for different problems.

1. Consider the following cruel and unusual sorting algorithm.

```
CRUEL(A[1 .. n]):
  if n > 1
      CRUEL(A[1 .. n/2])
      CRUEL(A[n/2 + 1 .. n])
      UNUSUAL(A[1 .. n])
```

```
UNUSUAL(A[1 .. n]):
  if n = 2
      if A[1] > A[2]                  ⟨⟨the only comparison!⟩⟩
          swap A[1] ⟷ A[2]
  else
      for i ← 1 to n/4                ⟨⟨swap 2nd and 3rd quarters⟩⟩
          swap A[i + n/4] ⟷ A[i + n/2]
      UNUSUAL(A[1 .. n/2])            ⟨⟨recurse on left half⟩⟩
      UNUSUAL(A[n/2 + 1 .. n])        ⟨⟨recurse on right half⟩⟩
      UNUSUAL(A[n/4 + 1 .. 3n/4])     ⟨⟨recurse on middle half⟩⟩
```

Notice that the comparisons performed by the algorithm do not depend at all on the values in the input array; such a sorting algorithm is called ***oblivious***. Assume for this problem that the input size n is always a power of 2.

(a) Prove that CRUEL correctly sorts any input array. *[Hint: Consider an array that contains $n/4$ 1s, $n/4$ 2s, $n/4$ 3s, and $n/4$ 4s. Why is considering this special case enough? What does* UNUSUAL *actually do?]*

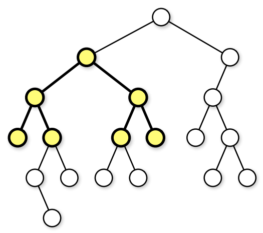
(b) Prove that CRUEL would *not* always sort correctly if we removed the for-loop from UNUSUAL.

(c) Prove that CRUEL would *not* always sort correctly if we swapped the last two lines of UNUSUAL.

(d) What is the running time of UNUSUAL? Justify your answer.

(e) What is the running time of CRUEL? Justify your answer.

2. For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. Describe and analyze a recursive algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return the root and the depth of this subtree.



The largest complete subtree of this binary tree has depth 2.

3. (a) Suppose we are given two sorted arrays $A[1..n]$ and $B[1..n]$. Describe an algorithm to find the median of the union of A and B in $O(\log n)$ time. Assume the arrays contain no duplicate elements.

   (b) Now suppose we are given *three* sorted arrays $A[1..n]$, $B[1..n]$, and $C[1..n]$. Describe an algorithm to find the median element of $A \cup B \cup C$ in $O(\log n)$ time.

*4. *Extra credit; due September 17.* (The "I don't know" rule does not apply to extra credit problems.)

Bob Ratenbur, a new student in CS 225, is trying to write code to perform preorder, inorder, and postorder traversal of binary trees. Bob understands the basic idea behind the traversal algorithms, but whenever he tries to implement them, he keeps mixing up the recursive calls. Five minutes before the deadline, Bob submitted code with the following structure:

```
PREORDER(v):
    if v = NULL
        return
    else
        print label(v)
        ███ORDER(left(v))
        ███ORDER(right(v))
```

```
INORDER(v):
    if v = NULL
        return
    else
        ███ORDER(left(v))
        print label(v)
        ███ORDER(right(v))
```

```
POSTORDER(v):
    if v = NULL
        return
    else
        ███ORDER(left(v))
        ███ORDER(right(v))
        print label(v)
```

Each ███ represents either PRE, IN, or POST. Moreover, each of the following function calls appears exactly once in Bob's submitted code:

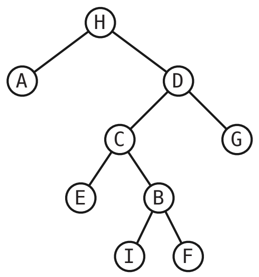
PREORDER(*left*(v))	INORDER(*left*(v))	POSTORDER(*left*(v))
PREORDER(*right*(v))	INORDER(*right*(v))	POSTORDER(*right*(v))

Thus, there are exactly 36 possibilities for Bob's code. Unfortunately, Bob accidentally deleted his source code after submitting the executable, so neither you nor he knows which functions were called where.

Your task is to reconstruct a binary tree T from the output of Bob's traversal algorithms, which has been helpfully parsed into three arrays $Pre[1..n]$, $In[1..n]$, and $Post[1..n]$. Your algorithm should return the unknown tree T. You may assume that the vertex labels of the unknown tree are distinct, and that every internal node has exactly two children. For example, given the input

$$Pre[1..n] = [\texttt{H A E C B I F G D}]$$
$$In[1..n] = [\texttt{A H D C E I F B G}]$$
$$Post[1..n] = [\texttt{A E I B F C D G H}]$$

your algorithm should return the following tree:



In general, the traversal sequences may not give you enough information to reconstruct Bob's code; however, to produce the example sequences above, Bob's code must look like this:

```
PreOrder(v):
  if v = Null
      return
  else
      print label(v)
      PreOrder(left(v))
      PostOrder(right(v))
```

```
InOrder(v):
  if v = Null
      return
  else
      PostOrder(left(v))
      print label(v)
      PreOrder(right(v))
```

```
PostOrder(v):
  if v = Null
      return
  else
      InOrder(left(v))
      InOrder(right(v))
      print label(v)
```